Divide and Conquer

How breaking down a problem helps solving it faster

The concept

- Break down the problem in smaller problems
- Solve the sub-problems
- Combine the results

When the problem reduces to a single subproblem, the term *decrease and conquer* is sometimes used

→ e.g binary search

Binary Search - A *Decrease and Conquer* algorithm

How do you find an item in a sorted list?

ME KNOWS!!

Me goes through all the list
until me finds the item

→ O(n)

Binary Search - A *Decrease and Conquer* algorithm

But ... the list is **sorted** (let's say in increasing order)

So if you look at any item in the list:

- if it's greater than the item you're searching, yours should be more on the left
- if it's smaller than the item you're searching, yours should be more on the right

1 2 7 13 17 29 42

if you're looking for 37, you should go on the right

Binary Search - A *Decrease and Conquer* algorithm

The binary search algorithm is the following:

- Check the element at the center of your list
- Compare it to the element you're searching
- Keep either the lower half or the upper half
- Reiterate until you find the element or your search range is empty

The search range is divided by 2 every iteration.

$\rightarrow O(\log_2 n)$

Note: $O(\log_2 n) \leftrightarrow O(\log n)$ because $\forall\ i, j > 1, \log_i n = \log_j n / \log_j i$

Binary Search - A *Decrease and Conquer* algorithm

```
arr = [int(e) for e in input().split()]
item = int(input())

start = 0
end = len(arr)

while end > start:
    mid = (end + start) // 2
    if arr[mid] == item:
        print("Found at index %d" % mid)
        break
    elif arr[mid] > item:
        end = mid
    else:
        start = mid + 1
else:
    print("Not found")
```

Looking for 8:

1 2 7 13 17 29 42 55 72 112 404 666
1 2 7 13 17 29 42 55 72 112 404 666
1 2 7 13 17 29 42 55 72 112 404 666
1 2 7 13 17 29 42 55 72 112 404 666
1 2 7 13 17 29 42 55 72 112 404 666
Not found

`else` can be used after a `while` or `for` loop. It is executed when the loop terminates normally (without a `break`)

Exponentiation by squaring - A *Decrease and Conquer* algorithm

How to compute a^n more efficiently than multiplying a by itself n-1 times?

```
def fast_exp(a, n):
    if n == 1:
        return a
    elif n % 2:
        return a * fast_exp(a, n - 1)
    else:
        return fast_exp(a, n // 2)**2
```

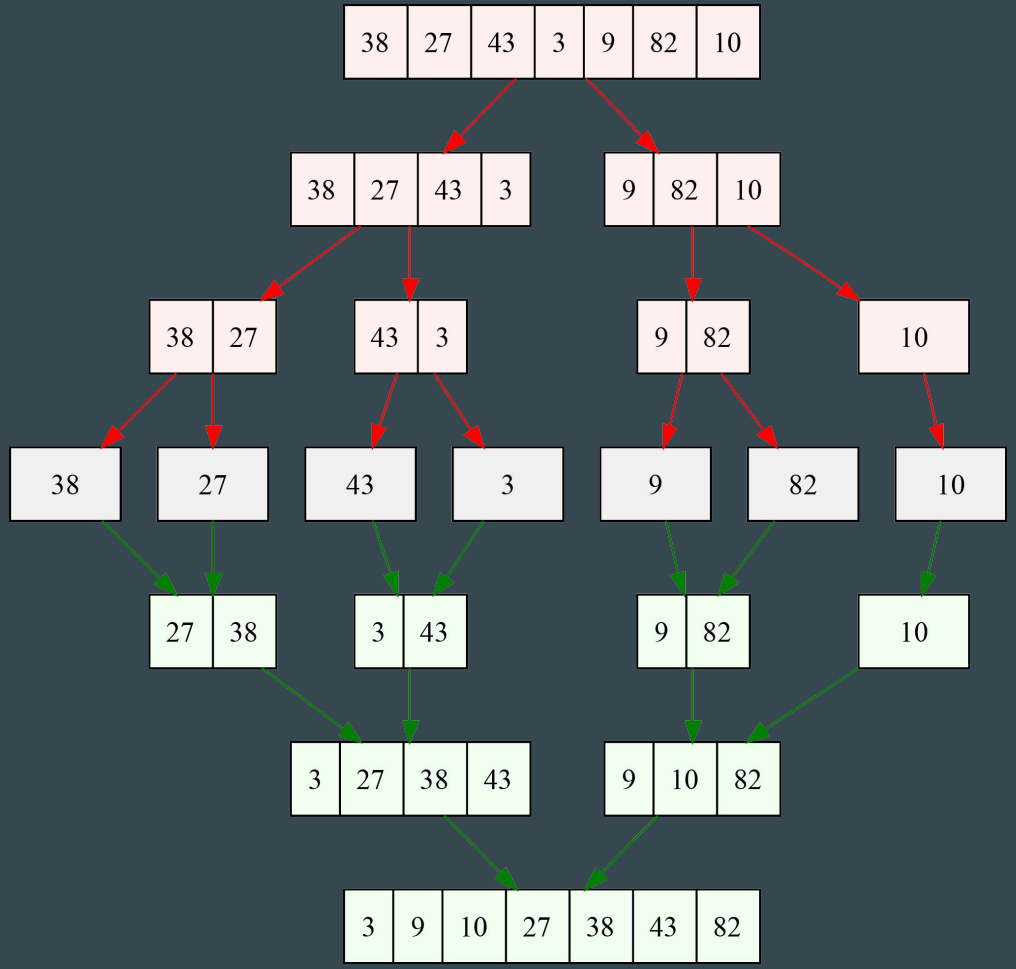
→ O(log n)

Merge sort

Method to sort a list

- Split the list in two
- Recursively sort the sub lists
- Combine the two sub lists:
  - check the first item of each list
  - take the smallest, remove it and put it in the new list
  - repeat the operation until you have taken all the items

→ O(n log n)



Merge sort

Exercise: implement a merge sort

1. Recursive version, top-down (a bit easier)
2. Iterative version, bottom-up (can sort in-place)

Top-down: split in two until the size of the sub-lists is one

Bottom-up: construct runs of size 1, 2, 4, ..., n

Credits

Slides: Louis Sugy for INSAlgo

Quick sort schema: Wikipedia

and me for
brilliant ideas